The following security alert was issued by the Information Security Division of the Mississippi Department of ITS and is intended for State government entities. The information may or may not be applicable to the general public and accordingly, the State does not warrant its use for any specific purposes.

DATE(S) ISSUED:
12/11/2009

SUBJECT:
New Reports of SQL Injection Attacks

OVERVIEW:
Recent SQL Injection attacks have underscored the continued prevalence of SQL injection attacks as a viable means of exploitation. The purpose of this bulletin is to focus attention on the basic methodology of SQL injection and how these attacks can be prevented.

SQL injection is an attack in which malicious SQL code is inserted into application inputs (variables, text boxes, URL parameters, etc.) and are later passed to an instance of an SQL server for parsing and execution.

RISK:

Government:
Large and medium government entities: **High**
Small government entities: **High**

Businesses:
Large and medium business entities: **High**
Small business entities: **High**

Home users: High

DESCRIPTION:
Beginning in late November 2009, several hundreds of thousands of web pages across the world have been victims of ongoing automated SQL injection attacks. The attacks use SQL injection to silently re-direct web clients from trusted (but compromised) web sites to third-party web sites hosting malicious JavaScript code. Based on our analysis when users visit a compromised Web page, the injected iframe attempts to visit the following malicious IP addresses over port 80/TCP:

174.139.5.69
121.12.116.32
119.42.225.184
119.42.227.250

It should be noted that this is not an all inclusive list and may change over time.

The attacker appears to be focusing on the following software vulnerabilities:

Integer overflow vulnerability in Adobe Flash Player, described in CVE-2007-0071
MDAC ADODB.Connection ActiveX vulnerability described in MS07-009
Microsoft Office Web Components vulnerabilities described in MS09-043
Microsoft video ActiveX vulnerability described in MS09-032
Internet Explorer Uninitialized Memory Corruption Vulnerability described in MS09-002

Successful exploit leads to the silent installation of Backdoor.Win32.Buzus.croo. Once on the system, the rootkit-enabled Buzus.croo places the following files into the specified folders:

%UserProfile%\ammxv.drv
%ProgramFiles%\Common Files\Syesm.exe

The trojan then modifies the Registry to load when Windows is started by adding the following registry keys:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\DrvKiller
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\DrvKiller\Security
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\DrvKiller
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\DrvKiller\Security

Backdoor.Win32.Buzus.croo then attempts to contact 121.14.136.5 via port 80/TCP and sends a POST request to hxxp://dns.winsdown.com.cn/Countdown/count.asp.

The Buzus family of trojans typically are remotely controlled via an IRC backdoor and typically are engaged in credit card and other banking-related theft.

The majority of antivirus vendors are detecting this malware as can be seen in the following Virustotal report:

http://www.virustotal.com/analisis/f7637523c5aa2c0c2ddcb8cbc895732ed4a9ca83976885c3d458350e1d203f2a-1260487319

General Strategies for SQL Injection Mitigation

The best way to protect your database applications from SQL injection is to ensure developers do not allow client-supplied data to modify SQL statement syntax. **All user-supplied data must be thoroughly validated** before it is sent to a backend database. Examples of input attributes which should be checked are type, length, and format.

There are typically two ways that a developer or administrator can prevent bad user input: either disallow bad characters or only allow required characters. The second option is generally considered to be more secure because it is possible that filters may fail due to new attack methods, different character encoding, and other attack variables. It is preferable to take the time

to determine exactly what input is needed and then to reject any incoming request that does not adhere to the requirements.

There are a number of online resources to assist developers in performing input sanitization. The Open Web Application Security Project (OWASP) Validation Project includes tools and guidance for a number of programming languages, including Java, .NET, and PHP. Additionally, the OWASP Stinger Project includes an input validation component for Java/J2EE applications. Please refer to the References below for links to these and other secure programming resources.

Additional best practices to avoid SQL injection include:

> Avoid using dynamic SQL whenever possible. Dynamic SQL refers to any situation in which user-supplied input is concatenated with pre-defined SQL. Stored procedures or parameterized SQL can be used as a safer alternative.
> Apply the principle of least-privilege to web applications that interact with your database. It is a good idea to create an account for your web applications which has as few data access rights as possible to limit the scope of damage in the event that a system is compromised.
> Turn off debugging information as it is often used to gather data for subsequent attacks.

RECOMMENDATIONS:
The following actions should be considered:

- Enable full logging on web and application servers, such that all HTTP request information (including POST data) is logged. This extended information helps to identify the commands the attackers are attempting to execute.
- Review all logs (web server, application, network and database) for any references to the domains mentioned above.
- Review web server contents for references to suspicious domains and the IP addresses mentioned above.
- Ensure that all anti-virus software is up to date with the latest signatures.
- Ensure that the latest updates are applied to critical systems and desktops.

Application developers should take the following actions:

> Review server applications for possible SQL injection vulnerabilities, and apply all necessary code revisions and appropriate vendor patches after appropriate testing.
> Perform the actions outlined in the *Strategies for Mitigation Section* above.

REFERENCES:

Net Security
http://www.net-security.org/secworld.php?id=8604

DarkReading

http://www.darkreading.com/vulnerability_management/security/attacks/showArticle.jhtml?articleID=222001558

Open Web Application Security Project (OWASP)
http://www.owasp.org/index.php/SQL_injection
http://www.owasp.org/index.php/Category:OWASP_Validation_Project
http://www.owasp.org/index.php/Category:OWASP_Stinger_Project

Web Application Security Consortium (WASC)
http://www.webappsec.org/projects/threat/classes/sql_injection.shtml

CERT
http://www.cert.org/tech_tips/malicious_code_mitigation.html

US-CERT
https://buildsecurityin.us-cert.gov/

Microsoft
http://blogs.iis.net/nazim/archive/2008/04/28/filtering-sql-injection-from-classic-asp.aspx
http://msdn.microsoft.com/en-us/library/bb671351.aspx
http://msdn.microsoft.com/en-us/library/aa302415.aspx

CodePlex
http://www.codeplex.com/IIS6SQLInjection

NGSSoftware
http://www.ngssoftware.com/papers/asp.pdf

SecurityFocus
http://www.securityfocus.com/infocus/1596

ShadowServer
http://www.shadowserver.org/wiki/pmwiki.php?n=Calendar.20080513
http://www.shadowserver.org/wiki/pmwiki.php?n=Calendar.20080507